面向多核众核处理器的并行视频编码研究

颜成钢 张勇东 代 锋 张 峻 刘炳涛

1 引言

为了满足对视频服务质量不断提高的需求，研究人员一直在研究发展先进的视频压缩编码技术，以持续提高视频压缩编码的率失真性能（rate-distortion performance）。在此背景下，各种视频编码标准不断地出现，从早期的 MPEG-1[1]、H.261[2]到后来的 MPEG-2/4[3]、H.263/4[4~5]，以及新兴的 HEVC/H.265[6]。随着视频编码标准的发展，编码算法的复杂度持续提升，从而导致其对计算能力的需求不断提高。目前传统的单核处理器已经很难满足当前先进的视频编码技术对计算能力的要求[7]。在这种情况下，面向多核(Multi-core)/众核(Many-core)处理器的并行视频编码，作为满足视频编码对计算能力需求的重要技术手段，已经成为视频编码领域的一个热点研究方向，具有重要的学术价值和巨大的应用前景。目前新兴的视频编码标准 HEVC 也开始引入面向多核处理器的并行方案，在标准中增加了几种适用于并行编码的规格[6]。

多核处理器与众核处理器的主要区别在于处理单元数目（核数），目前一般认为单个芯片处理单元数目在 16 个以下的为多核处理器，超过 16 个为众核处理器。多核/众核结构的并行处理器正在高速发展。单个芯片上可容纳的处理单元数目约每隔 18 个月便会增加一倍。预计到 2018 年，单个芯片上的处理单元数目将达到 1000 个以上。我们已经步入或者即将进入众核处理器时代[8, 9]。

众核处理器时代的到来，为视频编码技术提供了强大的计算能力保障，但是也给视频编码领域研究带来了极大的挑战[7,10]。为了发挥众核处理器的并行计算能力，必须有与之相适配的高并行度视频编码技术与方法。而已有的并行视频编码方法主要面向多核处理器，其中大部分方法并行度不高，无法充分挖掘众核处理器的计算能力。研究适用于众核处理器的高并行度视频编码方法，为视频编码发展提供持续的计算能力保证，具有重要意义。目前面向众核处理器的高并行度视频编码研究刚起步，相关研究成果还比较少。

2 国内外研究概况

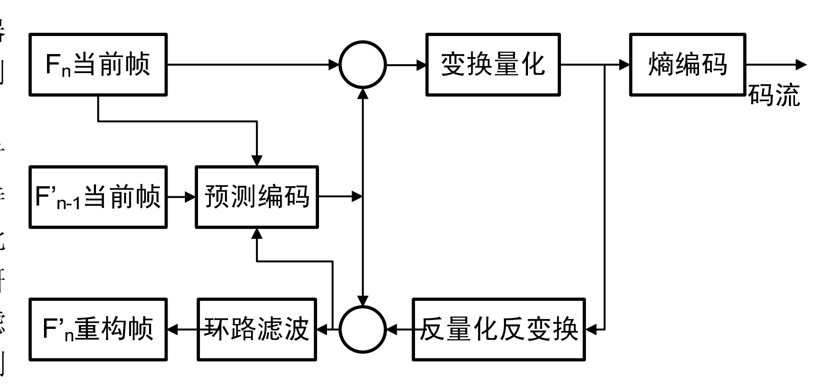


图1. 视频编码器的一般框架

图 1 是视频编码器的一般框架。其中预测编码特别是运动估计，是单核处理器上占用时间比例最大的部分，特别需要并行处理，因此关于并行运动估计的研究比较多[11~13]。环路滤波和熵编码，由于控制密集和与数据强相关，不容易实现并行。同时，由于其他模块并行化的发展，环路滤波和熵编码所占的时间比例也越来越大，开始成为性能

的瓶颈[10,14]。面对众核时代的到来，我们急需针对熵编码和环路滤波的并行方案。目前新兴的视频编码标准 HEVC 也开始考虑采纳针对这三个模块的并行方案[6]。我们重点针对这三个模块，开展了面向众核处理器的并行化方法，包括：面向众核处理器的并行环路滤波、面向众核处理器的并行熵编码、面向众核处理器的并行运动估计这三个方面的研究。这三个模块以外的变换量化和反量化反变换模块，因为容易实现高度并行化，同时在整个编码过程中所占的时间比例比较少，不是我们的研究重点。

2.1 并行环路滤波研究现状

并行环路滤波研究起步很晚，文献[15]最先于 2009 年提出了面向多核的并行滤波方案。所有研究者都集中于数据级的并行，主要包括宏块级并行[16~18]和像素级并行[15]。以下我们进行简单的介绍：

2.1.1 宏块级并行

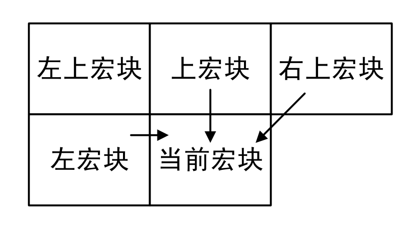


图2. 相邻宏块的相关性

1	2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16	17
11	12	13	14	15	16	17	18	19
13	14	15	16	17	18	19	20	21
15	16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24	25

图3. 2D-wavefront 宏块级并行方法

如图 2 所示，在环路滤波过程中，当前的宏块和临近的左、上和右上三个临近宏块相关。在对当前宏块滤波前，需要对这三个相关宏块滤波。如图 3 所示，目前存在的 2D-wavefront（二维波前）宏块级并行方案[16~18]直接将不相关的宏块分配给不同的处理单元进行滤波。每个矩形代表帧图像中的宏块，数字代表时间戳，具有相同数字的宏块可以并行处理。

我们发现，刚开始滤波的时候，每过两个时间戳，并行度加一。如果处理单元数目足够多，帧图像水平方向和垂直方向的宏块数目分别为 $\boldsymbol{W_m}$ 和 $\boldsymbol{H_m}$，则最大并行度为 $\mathbf{min}(ceil(W_m/2), H_m)$。这样的并行度还远不能充分利用众核处理器所有的处理单元；同时，每个宏块处理之前，需要和三个宏块进行同步通信。如果相关宏块没有处理完毕，该宏块需要继续等待。整个帧图像的同步通信次数近似于 $3\times \boldsymbol{W_m}\times \boldsymbol{H_m}$。频繁的宏块间同步通信产生了大量的等待时间，严重影响整体性能，同步负载很重；而且，因为滤波强度不尽相同，各个宏块的滤波时间可能不一样。滤波时间短的处理单元需要等待滤波时间长的处理单元，负载不均衡问题严重。

2.1.2 像素级并行

如图 4 所示，文献[15]首先分析滤波像素和相关像素之间的关系。宏块滤波边界 W 可能会影响像素 b~g 的数值，在对像素 j 进行滤波的时候，需要的相关像素为 h~k。j 的相关像素和 W 的滤波像素不重合，j 右边的像素也是如此。这样在对边界 W 滤波的同时，可以对像素 j 以及 j 右边像素进行滤波。图 5 描述了文献[15]的像素级并行方案，将帧图像分成很多像素块，分别对其并行处理。此方法的优点很明显：一个帧图像可以分成很多无相关性的像素块，并行度大，同时各个像素块之间没有通信，同步负载小。最大的问题是，文献[15]对滤波像素和相关像素的相关性分析忽略了编码标准中的一些限制因素，这会严重影响编码

效率。另外，每个像素块的处理时间不一样，存在严重的负载不均衡问题。

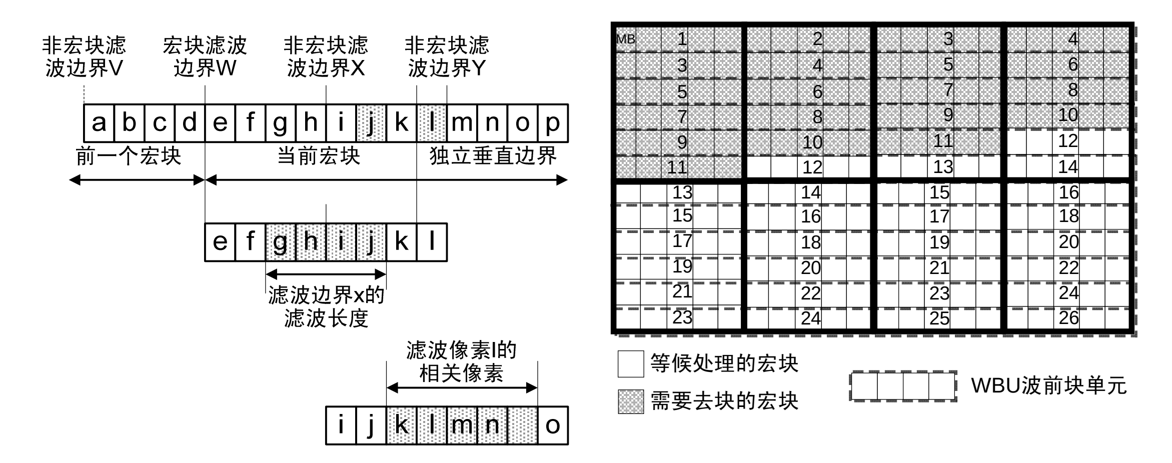


图4.　滤波像素和相关像素的关系[15]　　　　图5.　像素级并行方案[15]

2.2 并行熵编码研究现状

熵编码主要包括变长熵编码和算术熵编码两种。并行熵编码研究起步也很晚，集中于算术熵编码的并行算法研究。文献[19]最先提出了面向多核的并行熵编码方案。到目前为止，并行熵编码方案主要有两种：一种是语法元素分割[19,20]，形成相互之间相关性比较小的语法元素集合，这些集合可以并行执行；另一种是熵编码片（entropy slice）分割，在熵编码阶段，将片(slice)分割成相互独立的熵编码片，这样就增加了熵编码的并行度[6]。

2.2.1 语法元素分割

表 1 和表 2 及图 6 分别是文献[19]和文献[25]的语法元素分割方法。以图 6 为例，在熵编码阶段，首先将语法元素分割成五个集合，对这五个集合分别进行熵编码，在熵解码阶段，再对这五个集合分别进行熵解码。这样就大大提高了熵编解码的性能。这五个集合之间相关性很弱，但还是存在一定的相关性。如果对这五个集合直接并行处理，势必会严重影响编码效率。同时，各个语法元素集合的处理时间不一样，存在负载不均衡问题。另外，表 1 和表 2 及图 6 的语法元素集合的个数分别是 3 和 5，对应的最大并行度只有 3 和 5，并行度太小。

表1.　文献[19]的语法元素分割方案

组别	语法元素特点
组 I	语法元素是关于帧头的信息，包括宏块类型、预测信息和解码控制信息、片尾
组 II	语法元素是关于残余数据和片尾的映射信息
组 III	语法元素是关于残余数据、片尾的级别信息

表2.　文献[25]的语法元素分割方案

组别	语法元素
组 I	mb_skip_flag、mb_type、sub_mb_type、mb_field_decoded_flag、end_of_slice_flag
组 II	prev_intra4×4_pred_mode_flag、rem_ intra4×4_pred_mode、prev_intra8×8_pred_mode_flag、rem_ intra8×8_pred_mode、intra_chroma_pred_mode、ref_idx_10、ref_idx_11、mvd_10、mvd_11
组 III	transform_size_8×8_flag、mb_qp_delta、coded_block_pattern、coded_block_flag
组 IV	significant_coeff_flag、last_significant_coeff_flag
组 V	Coeff_abs_level_minus1、coeff_sign_flag

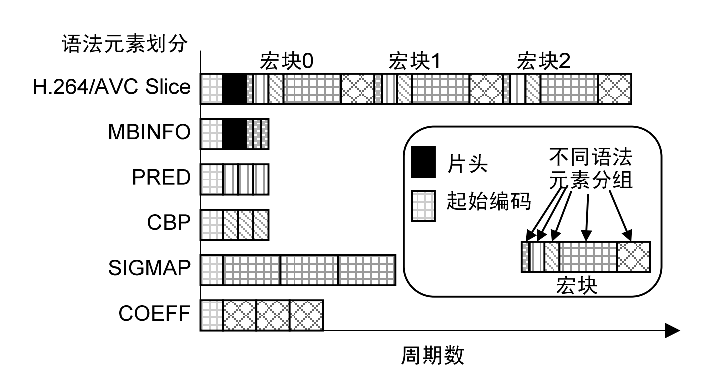


图6. 在该分割方案下的并发情况

2.2.2 熵编码片分割

在文献[21]的片(slice)级并行方案中，随着处理单元数目的增多，为了满足高并行度，片的数目也随即增多，这样会严重影响编码的效率。目前新兴的视频编码标准 HEVC 开始考虑采纳熵编码片(entropy slice)分割[6]。在熵编码阶段，将每个片进行分割，形成多个相互独立的熵编码片。如图 7 是将一个片分割成了三个熵编码片。在除熵编码以外的其他视频编码阶段，就没有熵编码片的概念了。所以其他阶段，片的数目没有增加，不会影响其他阶段的编码效率。这些熵编码片可以独立地进行熵编码，这样就增加了熵编码阶段的并行度，提高了速度。不过由于熵编码片之间的信息不能相互参考，这个方法也会一定程度影响熵编码的编码效率。同时由于熵编码片的处理时间不同，负载不均衡问题严重。

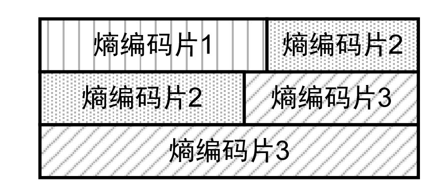


图7. 将一个片分割成多个熵编码片

2.3 并行运动估计研究现状

运动估计(motion estimation, ME)计算量巨大，在整个编码过程中占用了最大的比例。在 H.264 的参考代码中，运动估计要占用大于 80%的计算时间。因此研究如何在保证编码效率的前提下在运动估计模块内部最大限度地去耦合，以实现运动估计高度并行化，十分有意义。

2.3.1 面向传统编码标准的全局并行

面向传统视频编码标准（如 H.264）的并行运动估计研究较对于其他模块的并行研究更多。这些研究大多是基于 GPU 平台的全局并行。文献[22]将一帧图像划分成 4×4 的独立图像块，对每个块在搜索范围内的每一个候选位置（candidate position）使用一个线程独立地计算匹配代价 SAD（Sum of Absolute Differences）。然后将每一个宏块内的所有 4×4 块进行组合，进而可以得到其他形状块在搜索范围内的所有匹配代价。对每个块选择最小的匹配代价，对应的位置偏移即为运动矢量。文献[23]的思想与文献[22]大致相同，不同的是该论文考虑到了运动矢量估计（motion vector prediction，MVP）对编码性能的影响，使用运动矢量估计对搜索范围初始化，一定程度上减小了质量的下降。这类方法实现起来简单，并行度高，但是没有充分考虑到相邻宏块之间的编码相关性，会对编码质量造成较大的影响。

2.3.2 面向 HEVC 的局部并行

在 HEVC 编码标准中，每一个编码单元(coding unit, CU)会被划分成若干个预测单元(prediction unit, PU)，预测单元是携带运动信息(参考帧，运动矢量)的最小单元。与 H.264/AVC 相比，HEVC 对预测单元的划分更加灵活（矩形、方形、对称、非对称）以更好地适应不

同形状的运动区域。另外，HEVC 允许预测单元采用 merge/skip（合并/滤过）模式，预测单元不必显式地传送运动矢量和参考帧信息，编码端选取与当前预测单元相邻的其他已编码预测单元形成一个候选列表(merge candidate list, MCL)，然后从该列表中通过计算匹配代价选择一个与当前预测单元运动信息最接近的预测单元，将它的运动参数作为当前预测单元的运动参数。在码流中只需要传送最匹配预测单元在候选列表中的索引即可。解码端用同样的方法构造出候选列表，根据解码得到的索引即可获取当前预测单元的运动参数。候选列表的构造依赖于 5 个空域相邻预测单元（A0, A1, B0, B1, B2）和 2 个时域相邻预测单元 (C, H)，如图 8 所示。HEVC 中运动矢量估计的计算也与周围预测单元有类似的依赖关系。

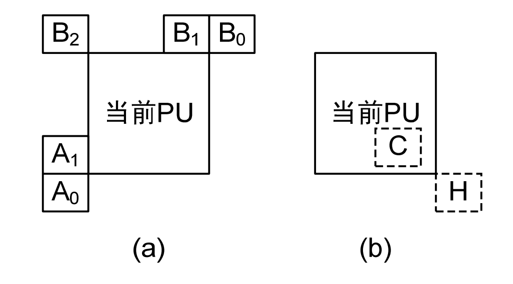


图8. 候选列表构造对相邻预测单元的依赖关系 (a)空域相邻预测单元, (b)时域相邻预测单元

HEVC 中 merge/skip 的引入和运动矢量估计的计算使预测单元之间的编码依赖性更强，不能再使用传统并行运动估计方法[22, 23]，否则会对编码质量造成很大的影响[24]。为了能在保证编码质量的前提下并行处理，提案[24]中提出了运动估计区域(motion estimation region, MER)的概念。每一个运动估计区域是最大编码单元（largest coding unit, LCU）的等分。所有运动估计区域均是正方形且尺寸相同。[24]认为同一个运动估计区域里面的预测单元无依赖关系，在每一个运动估计区域范围内所有的预测单元可以并行地进行运动估计，如图 9 所示。[24]虽然在一个运动估计区域内部对所有预测单元并行运动估计，但是运动估计区域之间仍然是按照扫描顺序进行处理，所以整体并行度不高。

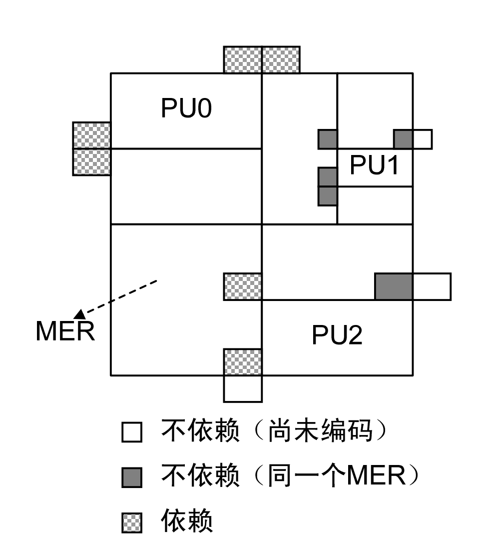


图9. 运动估计区域对周围预测单元的依赖关系的影响

3 X.264 视频编码器在多核平台上的并行化

x.264 是应用最为广泛的开源 H.264 编码器，我们对其在多核平台上进行了并行化性能的实验研究，通过实验数据分析算法的并行化能力。

3.1 实验配置

实验中采用了如下三种硬件平台：

– HW0：曙光 i950 服务器，含 8 个 Intel Xeon X7550@ 2GHz CPU。每个 CPU 为 8 核，共享 18MB 三级缓存（L3 cache)。
– HW1：华硕 P6T7 主板搭载一颗 Intel Core i7 950 @3.07GHz CPU。该 CPU 含有 4 核，共享 8MB 三级缓存 ，支持 Hyperthreading 技术，可同时运行 8 个线程。
– HW2：华硕 UX30 笔记本，搭载超低电压版 Intel SU7300 1.3GHz CPU，该 CPU 含

有 2 核，共享 3MB 的二级缓存。

实验采用了来自并行基准程序集 PARSEC 2.1 的 x264 作为研究对象，其输入为 1920x1080 分辨率、25fps、共 512 帧的 YUV[1]视频，时长 20.5 秒。实验中区分了如下两种实现选项：

- SW0：汇编优化
- SW1：无汇编优化

实验中采用的分析工具有：

- GP：gprof
- OP：oprofile
- TP：Agner Fog 的 testp 工具

3.2 实验一：线程数与性能的关系

基于硬件平台 HW0 和软件选项 SW0，通过改变线程数，观察实际运行时间与总耗时[2]。三个测得数据依次为实际运行时间、总耗时、加速比。加速比由总耗时除以实际运行时间得到。

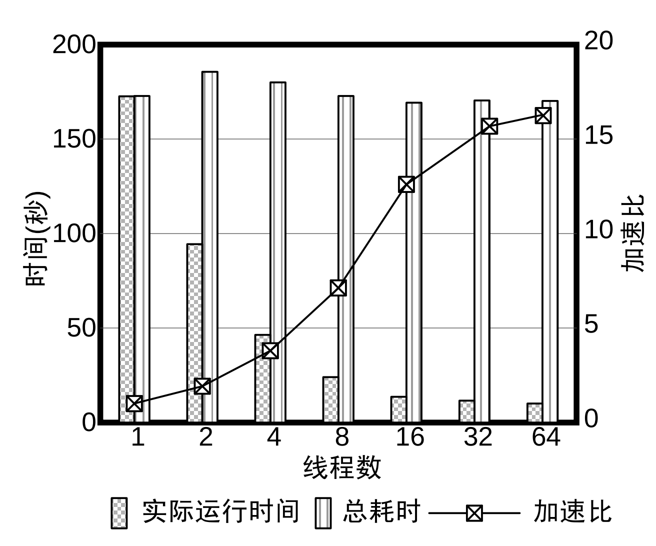


图10. 并行性能与线程数的关系

实验结果如图 10 所示。从中可以看出：

- 随着线程数增加，运行时间按比例减少，线程数为 2 和 4 时加速比约 2 和 4。线程数为 8 时，加速比为 7，加速比降低的原因是：加上非计算线程，总线程数超过 8，而代码中没有进行线程与处理器的绑定，部分线程被分配到不同的 CPU 执行(CPU 8 核)，不共享三级缓存的线程间通信代价增高。
- 16 线程加速比为 12.6，32 线程加速比为 15.7，64 线程为 16.1。更多的处理器加入运算，由于不共享三级缓存，基于总线的交互增多，通信计算比上升，效率下降。
- 预计使用更多的处理器，加速比存在极大值。原因有两个方面，一个是按照阿姆达尔（Amdahl）定律，串行部分所占比率决定了最大加速比；二是能同时编码的帧的数目有上限，限制了帧间的并行度。

3.3 实验二：汇编 SIMD 指令优化的加速效果

现代处理器中的 SIMD 向量部件对图像处理应用有较强的加速效果。本实验基于 HW1 平台，观察在采用和不采用汇编 SIMD 指令时的性能状况。实验中统计的线程数分别为 1、2、4，具体实验结果（实际运行时间与总耗时）如图 11 所示。其中，前两项为有汇编优化的情况下的运行时间，后两项为无汇编优化的情况下的运行时间。从试验结果可以看到：

[1] 一种颜色编码方法

[2] 实际运行时间指在现实中程序运行经过的时间；总耗时指所有核的运行时间之和。

– 随线程数增加，总耗时有增长，因为并行化需要额外开销。这个现象在 HW1 平台下比 HW0 平台下更明显，因为 x7550 xeon 与 i7 950 相比，三级缓存更大，运行频率较低，使得通信与计算能力比值更大。
– 即使线程数不同或者硬件平台不同，SIMD 指令的数据并行带来的加速比基本相同。
– 数据并行效果优于线程级并行。经汇编优化的单线程程序的实际运行时间少于未经汇编优化的四线程程序。不过两种加速方法可以同时使用，并不互斥。
– 数据并行使用硬件加速逻辑，虽然应用范围有限，但对适合的应用效果很好。

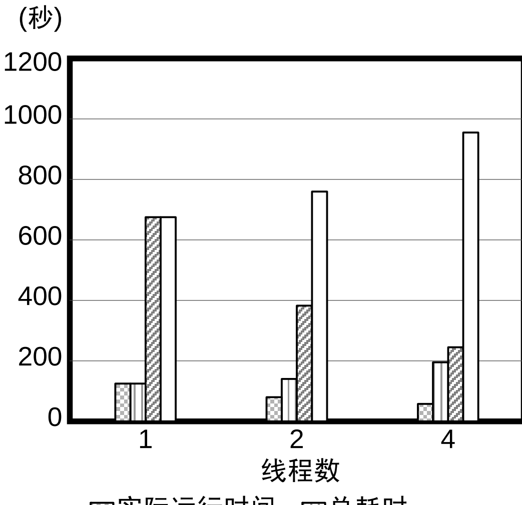


图11. 是否采用 SIMD 优化的效果

3.4 实验三：程序热点与调用图分析

我们基于硬件平台 HW1 和 HW2，采用 GP 和 OP 两种分析工具，针对有无汇编优化这两种不同情况，用 gprof 分析函数调用关系，用 oprofile 分析函数运行时间与程序热点。实验结果如表 3 到表 6 所示。

表3. 无汇编优化时的 gprof 分析结果

时间（%）	累计秒数	自身占用秒数	调用次数	函数名
51.55	315.89	315.89	352855786	x264_pixel_sad_x4_16x16
8.22	366.25	50.36	95480404	x264_pixel_satd_16x16
6.95	408.86	42.61	162003947	x264_pixel_sad_16x16
5.17	440.56	31.70	221451763	get_ref
2.70	457.07	16.52	79657183	mc_chroma
2.51	472.43	15.36	23183864	x264_pixel_sad_x3_16x16

表4. 无汇编优化时的 oprofile 分析结果
CPU: Intel Core/i7, 估计运行时频率 1600 MHz

CLK_UNHALTED		**LLC_MISSES**		**LLC_REFS**		二进制文件	函数符号名
样本点数	百分比	样本点数	百分比	样本点数	百分比		
753864	46.74	175	40.51	1310	20.08	x264	x264_pixel_sad_x4_16x16
133229	8.26	0	0.00	82	1.26	x264	x264_pixel_satd_16x16
124814	7.74	31	7.18	850	13.03	libc-2.11.3.so	/lib/libc-2.11.3.so
104249	6.46	71	16.44	862	13.21	x264	x264_pixel_sad_16x16
78652	4.88	49	11.34	691	10.59	x264	get_ref
43875	2.72	33	7.64	441	6.76	x264	mc_chroma
34554	2.14	0	0.00	21	0.32	x264	x264_pixel_sad_x3_16x16

表5. 有汇编优化时的 gprof 分析结果

时间（%）	累计秒数	自身占用秒数	调用次数	函数名
23.95	25.48	25.48		x264_pixel_sad_x4_16x16_sse2
7.54	33.50	8.02		x264_pixel_satd_8x8_internal_ssse3_phadd
6.40	40.31	6.81	20359200	x264_me_search_ref
5.87	46.55	6.24		x264_pixel_avg2_w16_sse2
5.25	52.14	5.59	63340684	block_residual_write_cabac
4.58	57.01	4.87		x264_pixel_sad_16x16_sse2

表6. 有汇编优化时的 oprofile 结果
CPU: Intel Core/i7，估计运行时频率 1600 MHz

CLK_UNHALTED		**LLC_MISSES**		**LLC_REFS**			
样本点数	百分比	样本点数	百分比	样本点数	百分比	二进制文件	函数符号名
77731	20.89	167	33.81	823	21.18	x264	x264_pixel_sad_x4_16x16_sse2
36983	9.94	19	3.85	286	7.36	libc-2.11.3.so	/lib/libc-2.11.3.so
25275	6.79	17	3.44	109	2.80	x264	x264_pixel_satd_8x8_internal_ssse3_phadd
22295	5.99	7	1.42	168	4.32	x264	x264_me_search_ref
20206	5.43	74	14.98	594	15.29	x264	x264_pixel_avg2_w16_sse2
18376	4.94	27	5.47	423	10.89	x264	x264_pixel_sad_16x16_sse2

通过试验结果可以看到：

- SIMD 汇编优化的代码三级缓存访问次数减少，所以 SIMD 指令提高了数据的利用效率，而未优化的代码因为要存储中间变量以及访存模式不规律导致数据利用率较低。
- SIMD 硬件加速的核函数运行时间缩短到原来的 1/10 左右，热点函数的时间占用比例下降，体现了明显的 SIMD 加速效果。

图 12 是在 HW2 平台下用 OP 对单线程无 SIMD 加速的程序分析所生成的调用图。每个圆圈代表一个函数，加黑字体为函数名，顶端为该函数运行占用总时间的百分比，函数名上方为调用该函数的函数所占百分比（总和为 100%），函数名下方为该函数调用的函数所占百分比（省略部分被调用函数，总和为 100%）。

从调用图可以看到，占用系统最多时间的函数有 SAD 和 SATD 运算等。x264_me_search_ref 是完成帧间编码的宏块运动估计的主要函数，占用总运行时间比例高达 69%。

x264_me_search_ref 以及其子函数调用占用总时间百分比由如下计算得到：

```
x264_me_search_ref        1.8%
refine_subpel            +0.35%×100%
x264_pixel_sad_x3_16x16  +2.1%×100%
x264_pixel_sad_16x16     +6.5%×100%
get_ref                  +4.5%×100%
```

```
x264_pixel_sad_x4_16x16   +44.9%×100%
mc_chroma                 +1.8%×80.6%
x264_pixel_satd_8x8       +1.9%×(72%+1.2%)
x264_pixel_satd_16x16     +6.9%×87%
                          =68.99%
```

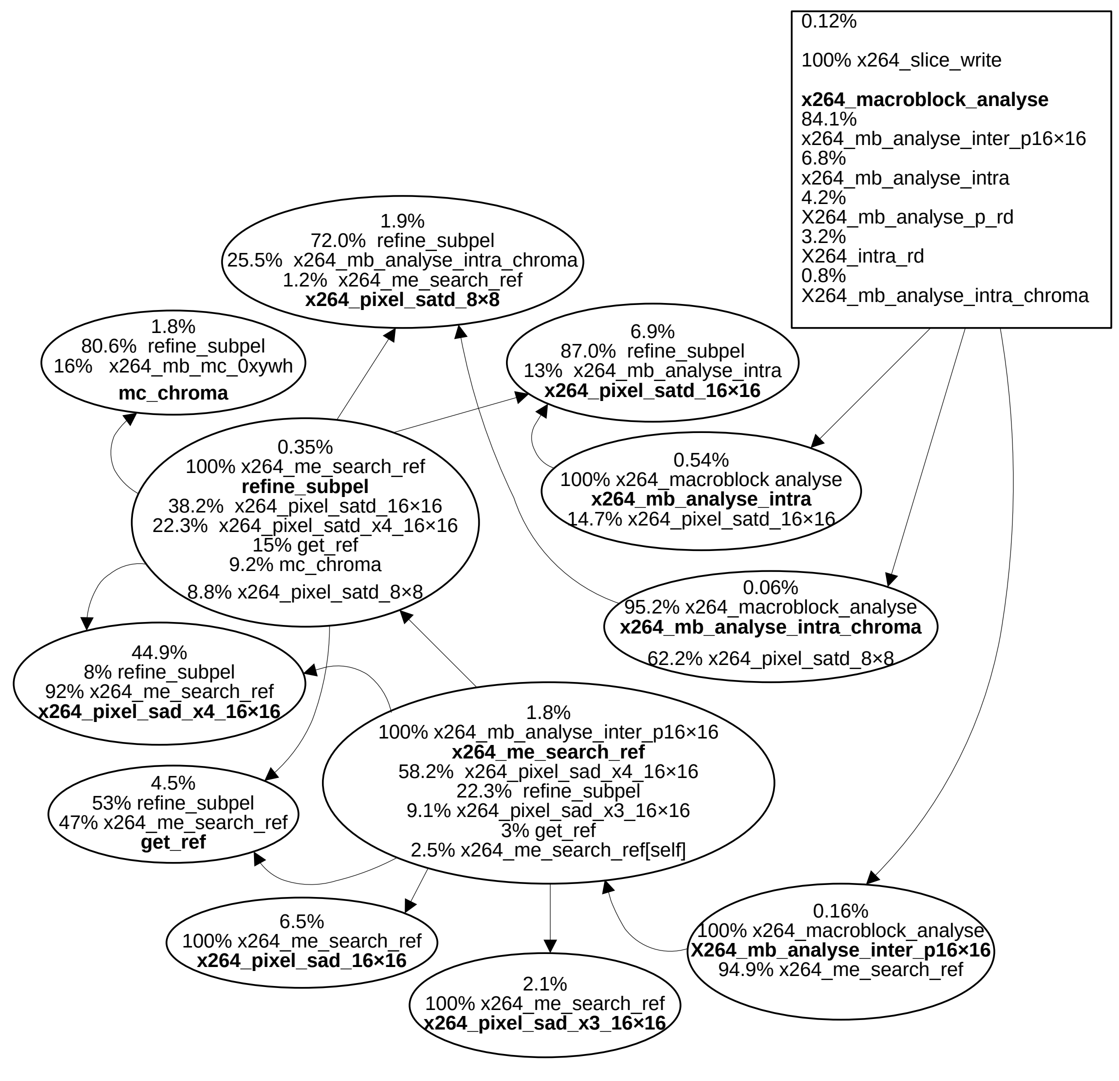


图12. x264 热点函数调用图

3.5 实验四：热点函数分析

Agner Fog 的 testp 工具可通过执行系统特权指令 rdtsc、rdpmc 等来读取硬件计时器和性能计数器，从而获得代码片段执行的统计信息，适合做详细的代码片段分析。本实验基于硬件平台 HW1 和 TP 工具，分别针对经过 SIMD 汇编指令优化和不经过优化的热点函数的代码片段，测量详细的执行周期数和微操作数等信息。

用于测试的函数代码片段主要实现 SAD 与 SATD 计算，根据其处理宏块的大小(4×4、8×8、16×16) 以及同时并行处理的宏块数(×1、×3、×4)区分为不同的函数。

试验结果如表 7 所示，左半部是没有进行汇编优化的代码的执行周期，右半部是进行汇编优化的同样功能的代码的执行周期。从中可以看出：

– 汇编优化后指令数减少。

– 数据越多，SIMD 的执行效率越高。表中显示：就执行效率而言，16×16×4 高于 16×16 高于 8×8。而没有汇编优化的代码，其执行时间与数据量呈基本固定的比例。
– sad_x4_16x16 的加速比达到 35 倍，最差的加速比达到 2.3 倍，考虑到各种核函数在运算中所占比重，这一结果与实验三热点函数的分析中加速一个量级的结果大体相符。
– 对于 SATD 中的阿达玛变换，没有专门硬件对该矩阵乘进行加速，所以加速比低于只需要加法操作的 SAD。说明适合硬件结构的运算能获得更好的加速效果。

通过分析热点函数的 c 语言实现，可以看到 SAD 与 SATD 都是非常简单的操作，仅涉及加减法、求绝对值、矩阵乘法等，其中应该还有进一步优化的空间。

表7. 热点函数代码片段分析

时钟周期	核内周期	指令数	微操作数	数据 L1 miss	时钟周期	核内周期	指令数	微操作数	数据 L1 miss
sad 16×16					sad 16×16 sse2				
1057	1098	3221	3203	0	51	48	74	100	0
sad 8x8					sad 8x8 mmxext				
281	289	843	781	0	28	26	44	62	0
sad 4×4					sad 4×4 mmxext				
80	71	226	208	0	14	14	28	42	0
satd 16×16					satd 8×8 ssse3_phadd				
1762	1835	4262	4571	0	241	249	460	611	0
satd 8×8					satd 8×8 ssse3_phadd				
451	464	1078	1162	0	77	79	128	174	0
satd 4×4					satd 4×4 ssse3				
103	110	262	283	0	45	42	81	91	0
sad 16×16×3					sad 16×16×3 sse2				
3145	3286	9657	9348	0	88	90	209	220	0
sad 16×16×4					sad 16×16×4 sse2				
4200	4376	12875	12540	0	120	122	266	279	0

4 面向众核处理器的 HEVC 并行编码

为了满足视频编码技术的发展对计算能力提出越来越高的要求，我们开展了面向众核处理器的高并行度视频编码关键技术的研究，重点研究适用于众核处理器的并行环路滤波、熵编码和运动估计方法，以有效解决现有方法并行度不足等问题，充分挖掘众核处理器的并行计算能力，为视频编码发展提供持续的计算能力保证。目前我们主要完成了面向众核处理器的并行环路滤波方法的研究，其他两部分简单介绍一下将来拟采用的方法。

4.1 面向众核处理器的并行环路滤波方法

目前的并行环路滤波方法都集中于数据级的并行，由于环路滤波控制指令非常密集，所以目前的方法存在并行度小、同步负载大、负载不均衡、影响编码效率的问题。我们引入任务级的并行方法，先将整个环路滤波过程分成两个任务，深入分析各个任务存在的问题，并提出了有针对性的解决方案。

4.1.1 分割成两个子任务----“边界强度计算”，“真假边界区分和滤波”

如图 13 所示，根据视频编码标准，环路滤波过程可以分成三个任务：滤波强度计算、真假边界区分、滤波。由它们的相关性可知，滤波强度计算与其他两个任务之间没有相关性，可以在进行真假边界区分和滤波之前，对所有的滤波强度先进行并行计算，这样就增加了并行度，提高了性能。

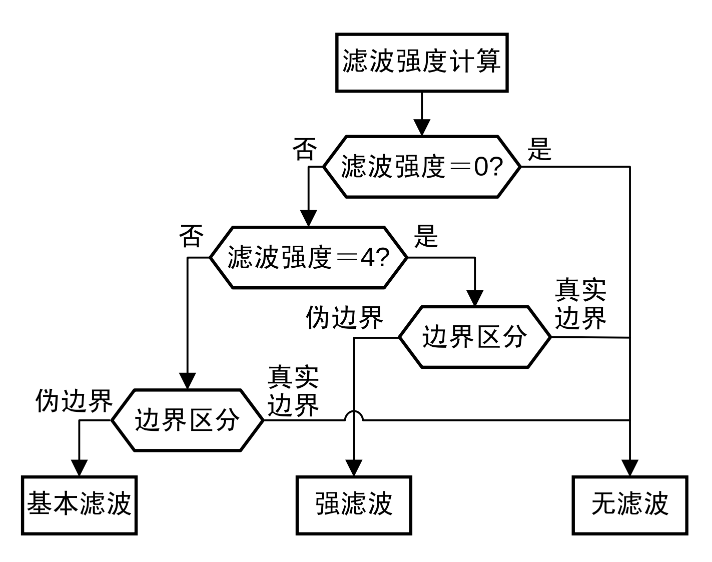


图13. 环路滤波三个任务之间的相关性

但是这也存在一定的问题：如图 14 所示，如果将滤波划分成两个任务，先并行执行任务“边界强度计算”，再对子任务“真假边界区分和滤波”采用常用的 2D-wavefront 方案，其中“BSC”代表子任务“边界强度计算”，“EDF”代表子任务“真假边界区分和滤波”。分割之后存在以下问题：

- 任务“边界强度计算”存在负载不均衡问题：如果我们将边界强度计算按照边界数量平均分给处理单元，每个处理单元分到的边界数量相同，但是每个边界强度计算的复杂度不同，各个处理单元负载之间会出现不均衡。
- 任务“真假边界区分和滤波”因为采用了 2D-wavefront 方法，并行度不够，同步负载开销很大。

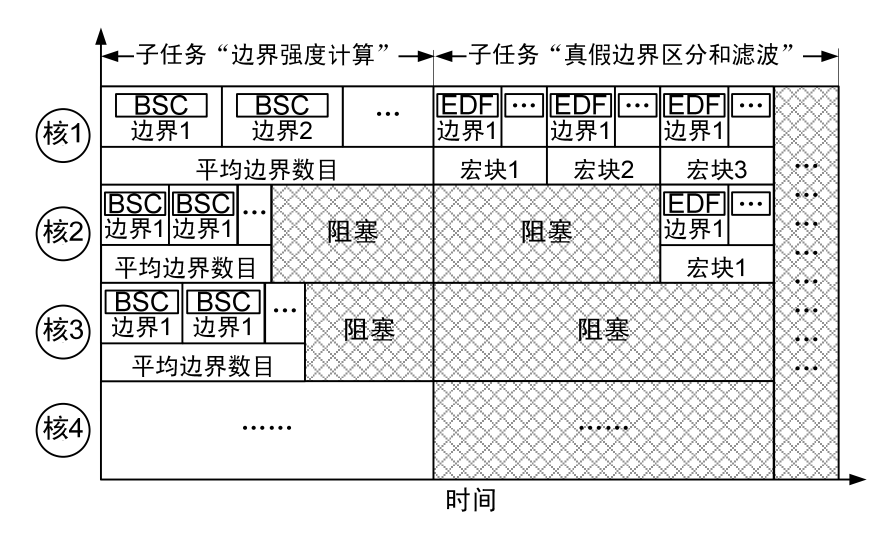


图14. 先对子任务“边界强度计算”并行执行，
再对子任务“真假边界区分和滤波”采用 2D-wavefront 方案

4.1.2 任务“边界强度计算”的并行加速方法-----马尔科夫转移概率加速方法

边界强度计算可以由公式（1）表示，其中：$\boldsymbol{T}$ 是边界强度计算决策树的各个分支运算，p_i 是选择第 i 个分支的概率，c_i 是第 i 个分支计算的复杂度。如果根据分支运算的概率分布，优先选择概率高的分支，边界强度计算的复杂度将会下降。

$$C(T)=\sum_{i\in\{0\cdots4\}} p_i \times c_i \tag{1}$$

通过仔细分析我们发现，对相同类型帧图像，当前边界强度与前一帧图像对应边界强度是相关的。视频编码标准中有三种帧类型，I、B 和 P 帧。我们以 P 帧为例，根据随机过程理论，P 帧的马尔科夫特性可以被描述为经验转移概率矩阵，我们描述经验转移概率矩阵如下：

滤波强度一共有五种状态，$S=\{s_0,s_1,s_2,s_3,s_4\}$，$s_0$、$s_1$、$s_2$、$s_3$、$s_4$ 分别代表：边界强度=0、边界强度=1、边界强度=2、边界强度=3、边界强度=4。假设当前滤波强度状态为 s_i，其下一帧滤波强度状态为 s_j 的概率为 p_{ij}，p_{ij} 就是转移概率，可以通过公式（2）获得。其中 n_{ij} 代表从状态 s_i 转移到状态 s_j 的个数，n 是所有转移的个数。公式（3）是转移概率矩阵，公式（4）和（5）是该转移概率矩阵的性质。

$$p_{ij}=\frac{n_{ij}}{n} \tag{2}$$

$$M=\left[p_{ij}\right] \tag{3}$$

$$p_{ij}\geq 0 \text{ for } 1\leq i,\ j\leq 5 \tag{4}$$

$$\sum_{j=1}^{5} p_{ij}=1 \text{ for } 1\leq i,\ j\leq 5 \tag{5}$$

我们从公共测试数据集 Xiph.org 中选取了几组训练数据，得到 P 帧图像的经验转移概率矩阵。利用这个经验转移概率矩阵，我们可以用霍夫曼树减小滤波强度计算，如公式（6），HT 是滤波强度分支计算的霍夫曼决策树。

$$C(HT)=\sum_{i\in leaf(HT)} p_i \times c_i \tag{6}$$

4.1.3 任务“真假边界区分以及滤波”的并行加速方法-----独立像素连通区域并行方法

任务“真假边界区分以及滤波”的数据相关性和整个环路滤波是一样的，以前的像素级并行方案忽略了编码标准的一些限制因素，在此我们对该任务进行像素级深入分析后，提出独立像素连通区域并行算法。

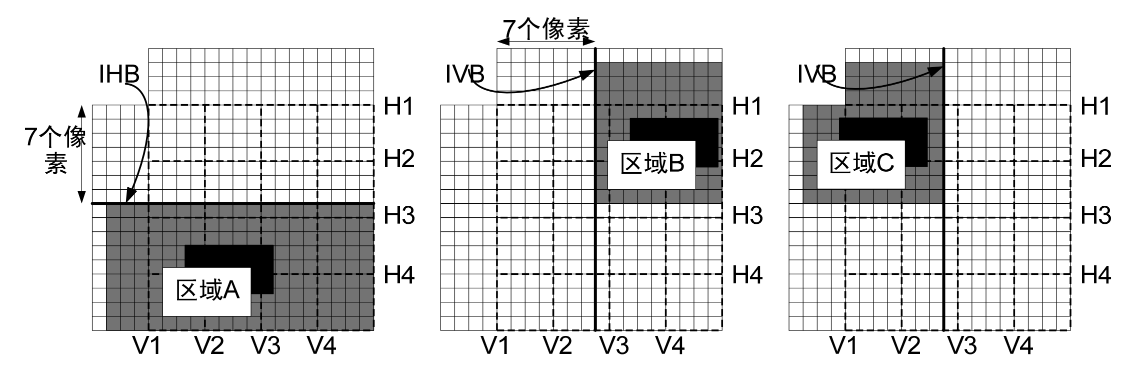


图15. 将每个宏块滤波边界影响的像素分成三个区域

首先分析滤波像素的相关像素和滤波边界的滤波长度之间的关系，这里我们讨论的是垂直滤波。如图 4(见§2·1·1)所示，滤波边界 X 的滤波长度为像素 g、h、i 和 j，而像素 l 在 Y 滤波中的相关像素为 k、l、m、n 和 o,所以和滤波边界 X 的滤波长度没有重合。因此进行 X

滤波之前，可以先对像素 l 进行 Y 滤波。像素 m、n、o 和 p 也可以在 X 滤波之前，进行 Y 滤波。我们将像素 k 和 l 的边界定义为独立垂直边界(IVB)。对水平滤波也会有类似的结果，我们也能找到独立水平边界(IHB)。

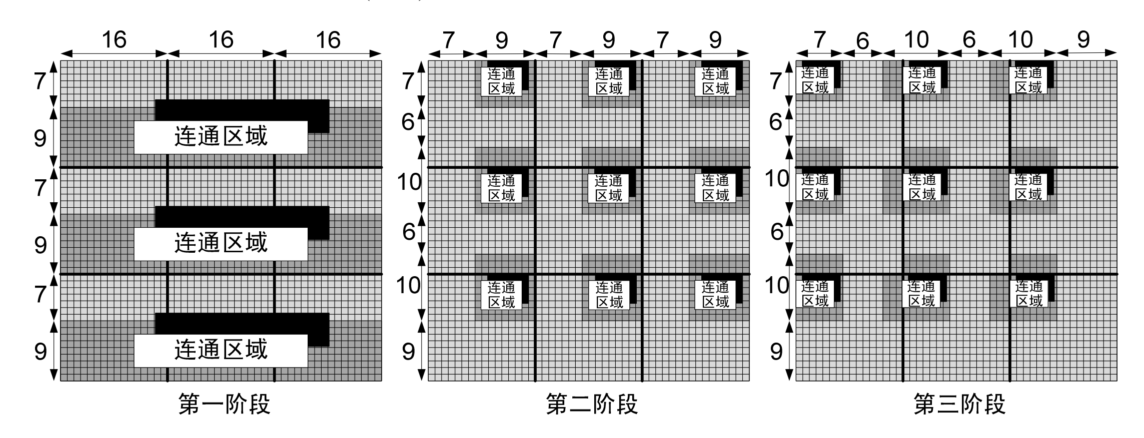


图16. 分成三个阶段对图像进行滤波

如图 15 所示，根据独立垂直边界 IVB 和独立水平边界 IHB，将受到每个宏块滤波边界影响的像素分成三个独立像素连通区域。每个连通区域的边界滤波是特定和有顺序的，这些顺序符合编码标准。如图 16 所示，我们将每帧图像的滤波过程分成三个阶段，_图像大小为 48×48，每个宏块大小为 16×16，每个宏块有一个 A、B 和 C，所以图像中一共有 9 个 A，B 和 C。在第一阶段中，每三个区域 A 形成一个连通区域，一共有三个连通区域。各连通区域平均分给处理单元进行处理。在第二和第三阶段中，每个连通区域只有一个区域 B 或者区域 C，一共有九个连同区域，也是平均分给处理单元进行处理。这三个阶段需要按次序进行，因为相互之间有相关性。我们发现，如果处理单元足够多，设帧图像水平方向和垂直方向的宏块数目分别为 $\boldsymbol{W_m}$ 和 $\boldsymbol{H_m}$，则刚开始滤波的时候，并行度就已经达到 $\boldsymbol{H_m}$，最大并行度可达 $\boldsymbol{W_m}\times\boldsymbol{H_m}$，并行度得到了很大地提高（如表 8）。同步通信只发生在相邻阶段之间，一共只有两次同步操作，同步负载比以前的方法明显减少。

表8. 不同方法的最大并行度对比

宏块数目	2D-wavefront	我们的方案
11×9（QCIF）	6	115
22×18（CIF）	11	430
45×36（SD）	23	1690
80×45（HD)	40	3688
120×68 (FHD)	60	8294

据我们实验估算，与流行的 2D-wavefront 方法对比，在保持编码效率不变的前提下，我们方法的加速比将达到 10 倍以上。

4.2 拟采用的并行熵编码方法

目前的并行熵编码方法主要有语法元素分割和熵编码片分割两种。语法元素分割方法影响编码效率，而且并行度太小。熵编码片分割方法会一定程度地提高并行度，但是会严重影响编码效率。我们将分别解决这两种方法存在的问题，并对解决方法进行融合，在保证编码效率的前提下提高并行度。

- **语法元素分割方法** 语法元素集合之间存在一定的相关性，如果直接对分割的语法元素集合并行处理，会影响编码效率。我们将深入分析语法元素集合之间的相关性，对分割的语法元素集合进行流水线处理，在保证编码效率的前提下，提高并行度。
- **熵编码片分割方法** 熵编码片之间不能相互参考，会影响编码效率。我们拟提出有限大小的熵编码片分割方法，并且选择最常用的上下文模型作为初始上下文模型，从而保证编码效率；熵编码片在除了熵编码以外的编码阶段，相互之间有相关性，

影响并行度。我们拟采用相隔熵编码片的组织方式，各个熵编码片在熵编码以外的阶段也可以并行处理，以在保证编码效率的同时，提升并行度。

– 由于语法元素分割方法和熵编码片分割方法可以同时存在，我们可以将它们进行结合，在保证编码效率的前提下提高熵编码的并行度。

4.3 拟采用的并行运动估计方法

在 HEVC 视频编码标准中，预测单元之间编码相关性很强，传统的并行运动估计方法不再适用。[24]提出的并行策略又有并行度不高的缺点。鉴于此情况，我们针对 HEVC 标准中的运动估计，研究适用于众核体系结构的高并行度运动估计算法，拟提出一种编码效率和并行度兼顾的预测单元级全局并行方法：

– HEVC 中 merge/skip 模式的候选列表构造对周围的预测单元有很强的依赖性。为了解除这种依赖性，我们使用 HEVC 提案 JCTVC-H0082[24]中提出的运动估计区域的概念，在一个运动估计区域范围内进行预测单元的并行运动估计。同时为了提高全局并行度，我们在此基础上设计了一种运动估计区域之间并行处理的方法，较提案[24]大大提高了并行度且保证了编码效率。

– 同样，HEVC 中运动矢量估计的计算对相邻预测单元的依赖性也很强，会阻碍并行化。但是不能忽略预测单元之间的依赖性，因为运动矢量估计对运动估计精确度有很大影响。因此为了保证编码效率，我们拟利用其他已编码帧中的运动矢量对内部预测单元的运动矢量估计进行预测，以避免运动矢量估计计算对并行度的影响。而边界预测单元的运动矢量估计可以精确得到，用得到的运动矢量估计进行搜索区域的初始化和匹配代价的计算，可以保证编码效率。

5 结束语

目前，我们提出的面向众核处理器的并行环路滤波方法已经在 2011 年度多媒体领域旗舰会议 ICME（国际多媒体会议及博览会，International Conference on Multimedia and Expo，2011）上发表，经过严格筛选，从 744 篇会议论文中脱颖而出，成为最佳论文候选。经过进一步完善，该项成果被推荐到多媒体领域顶级期刊 IEEE Trans. on Multimedia 上，已于该期刊的 2012 年 6 月期上发表。我们将对拟采用的并行熵编码方法和并行运动估计方法进一步开展研究，验证其有效性。 最后构建一个面向众核处理器的并行视频编码原型系统，以有效解决现有方法并行度不足等问题，充分挖掘众核处理器的并行计算能力，为视频编码发展提供持续的计算能力保证。

参考文献：

[1] ISO/IEC JTC1. ISO/IEC 11172, (MPEG 1) Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s Part 2: Video[S]. 1993.

[2] ITU-T. ITU-T Rec. H.261 Video codec for audiovisual services at PX64 Kbits/s[S]. 1990, 1; 1993, 2.

[3] ISO/IEC JTC1. ISO/IEC 14496-2 (MPEG-4 Part 2) Coding of audio-visual objects-Part 2: Visual[S], 1999.

[4] ITU-T. ITU-T Rec. H.263 Video coding for low bit rate communication[S]. 1995, 1; 1998, 2; 2000, 3.

[5] Joint Video Team of ITU-T and ISO/IEC JTC1. (ITU-T RecH.264 | ISO/IEC 14496-10 AVC) JVT-G050r1 Draft ITU-T recommendation and final draft international standard of joint video specification[S]. 2003.

[6] Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, High efficiency video coding (HEVC) text specification draft 6, Geneva, CH, 21–30 November, 2011.

[7] Rainer Leupers, LievenEeckhout, Virtual Manycore Platforms: Moving Towards 100+ Processor Cores. Design, Automation & Test in Europe Conference & Exhibition (DATE2011), Grenoble, 2011.

[8] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K. Arzen, VRomero, C. Scordino, Resource

Management on Multi-core Systems: the ACTORS Approach. Proc. IEEE International Symposium on Micro architecture, 2011, 31(3):72-81.

[9] N. Madan, A. Buyuktosunoglu, P. Bose, M. Annavaram, A case for guarded power gating in multi-core processors. The 17th IEEE International Symposium on High Performance Computer Architecture (HPCA-17), San Antonio, Texas, 2011.

[10] Ngai-Man Cheung, Xiaopeng Fan, Oscar C. Au, Man-Cheung Kung, Video coding on multi-core graphics processors, Proc. IEEE Signal Processing Magazine, 2010, 27(2):79-89.

[11] Erich Marth, and Guillermo Marcus, Parallelization of the x264 encoder using OpenCL, The 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques(SIGGRAPH 2010), Los Angeles, 2010.

[12] Javier Taibo, Víctor M. Gulías, Pablo Montero, Samuel Rivas, GPU-based fast motion estimation for on-the-fly encoding of computer-generated video streams, The 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video Vancouver(NOSSDAV 2011), British Columbia, Canada, 2011.

[13] Schwalb,M., Ewerth, R., Freisleben, B., Fast motion estimation on graphics hardware for h.264 video encoding, IEEE. Trans on Multimedia, 2009, 11(1):1-10.

[14] Chi Ching Chi, Ben Juurlink, A QHD-capable parallel H.264 decoder, Proceedings of the international conference on Supercomputing(ICS2011), Venice, Italy, 2011.

[15] Sung-Wen Wang, Shu-Sian Yang, Hong-Ming Chen, Chia-Lin Yang, Ja-Ling Wu, A multi-core architecture based parallel framework for H.264/AVC deblocking filters, J. Signal Process. Syst., 2009, 57(2):195-211.

[16] Chi Ching Chi, Ben Juurlink, Cor Meenderinck, Evaluation of parallel H.264 decoding strategies for the cell broadband engine, Proc. International Conference on Supercomputing(ICS2010), Venice, Italy, 2010.

[17] J.-Y. Lee , J.-J. Lee , S.M. Park, Multi-core platform for an efficient H.264 andVC-1 video decoding based on macroblockrow-level parallelism, IET Circuits, Devices & Systems, 2010, 4(2):147-158.

[18] Chenggang Yan, Feng Dai, Yongdong Zhang, Parallel deblocking filter for H.264/AVC on TILERA many-core systems, Proc. International Conference on Multimedia Modeling(MMM2011), Taipei, Taiwan, 2011.

[19] S. Chen, S. Chen, S. Sun, P3- CABAC: A nonstandard tri-thread parallel evolution of CABAC in the manycore era, IEEE Trans on Circuits and Systems for Video Technology, 2010, 20(6):920-924.

[20] Won-Jin Kim, Keol Cho, Ki-Seok Chung, Stage-based frame-partitioned parallelization of H.264/AVC decoding, IEEE Trans on Consumer Electronics, 2010, 56(2):1088-1096.

[21] Vivienne Sze, Anantha P. Chandrakasan, A Highly Parallel and Scalable CABAC Decoder for Next Generation Video Coding, IEEE J. Solid-State Circuits, 2012, 47(1):8-22.

[22] Ngai-Man Cheung, Xiaopeng Fan, Oscar C. Au, Man-Cheung Kung, Video Coding on Multicore Graphics Processors, IEEE Signal Processing Magazine, 2010, 27(2):79-89.

[23] Rodriguez-Sanchez, R., Martinez, J.L., Fernandez-Escribano, G., Claver, J.M., Sanchez, J.L., Reducing complexity in H.264/AVC motion estimation by using a GPU, Multimedia Signal Processing (MMSP2011), Hangzhou, 2011.

[24] Minhua Zhou, AHG10: Configurable and CU-group level parallel merge/skip, 8th Meeting of JCTVC, San Jose, CA, USA, 1-10 February, 2012.

[25] Vivienne Sze and Anantha P. Chandrakasan，Joint Algorithm-Architecture Optimization of CABAC，J Sign Process Syst (2012) 69:239–252

作者简介：

颜成钢： 中科院计算技术研究所前瞻研究实验室，北京市移动计算与新型终端重点实验室，博士生 yanchenggang@ict.ac.cn

张勇东： 中科院计算技术研究所前瞻研究实验室，北京市移动计算与新型终端重点实验室，研究员

代　锋： 中科院计算技术研究所前瞻研究实验室，北京市移动计算与新型终端重点实验室，副研究员

张　峻： 中科院计算技术研究所前瞻研究实验室，北京市移动计算与新型终端重点实验室，博士生

刘炳涛： 中科院计算技术研究所系统结构重点实验室，博士生 liubingtao@ict.ac.cn